Fault Grading of Software-Based Self-Test Procedures for Dependable Automotive Applications

P. Bernardi, M. Grosso, E. Sanchez
Dipartimento di Automatica e Informatica
Politecnico di Torino – Torino, Italy
{paolo.bernardi, michelangelo.grosso, ernesto.sanchez}@polito.it

O. Ballan
STMicroelectronics
Agrate Brianza – Milano, Italy
oscar.ballan@st.com

***Abstract*—Today, electronic devices are increasingly employed in different fields, including safety- and mission-critical applications, where the quality of the product is an essential requirement. In the automotive field, on-line self-test is a dependability technique currently demanded by emerging industrial standards. This paper presents an approach employed by STMicroelectronics for evaluating, or grading, the effectiveness of Software-Based Self-Test (SBST) procedures used for on-line testing microcontrollers to be included in safety-critical vehicle parts, such as in airbags and steering systems.**

Keywords-SoC, test, software-based self-test, fault grading

I. INTRODUCTION

In order to maintain competitiveness in the market, car manufacturers are constantly increasing their demand for automated features and performance in automotive electronics. In the last years, electronic control systems have begun to be employed in safety-critical areas, raising the need for total dependability. Different methodologies have been employed for assuring the computation correctness of the systems, addressing permanent and transient faults. They are generally based on different redundancy schemes or self-checking techniques, and may introduce significant costs in terms of engineering effort, silicon area and performance overhead. Depending on the specific application requirements and the statistical incidence of possible catastrophic faults, effective trade-offs have to be found between the level of fault detection or fault tolerance abilities and admissible costs. In the automotive field, a technique that is today demanded by emerging industrial standards such as [1] is the execution of self-test procedures at vehicle key-on, relying on hardware-based (e.g., Built-In Self-Test or BIST) or software-based techniques, such as SBST [2]. The latter option is based on running a suitable program by an available microprocessor, by means of which the correct behavior of the device is checked. This methodology is desirable due to its lower implementation costs, not requiring any system design modification.

This paper describes the current SBST strategy employed by STMicroelectronics for on-line checking the correct functionality of Systems-on-Chip (SoCs) included in safety-critical vehicle parts such as airbag and steering control systems. In particular, we describe the method used to evaluate the effectiveness of the SBST procedure provided by the designers and devised to guarantee a sufficient level of dependability. Results are shown for a SoC to be part of the airbag electronic control, which includes a processor connected to several peripheral cores.

978-3-9810801-7-9/DATE11/©2011 EDAA

II. SBST FOR IN-FIELD TESTING

Most of current digital designs include scan chains, but there are situations when their use is impossible or ineffective. This is the case of in-field testing, where there are no resources available to apply a scan-based test procedure. Other Design-for-Testability (DfT) approaches are quite common, such as BIST, where the test pattern generator and the response evaluator are integrated on-chip. This can be very effective and efficient, but circuit architecture modifications are required, which can introduce additional design and manufacturing costs, e.g., area overhead and speed penalties. SBST is an alternative solution which may overcome some of the limitations of DfT-based techniques in on-line testing of microprocessor-based systems. SBST techniques exploit the onboard microprocessor core to test itself, as well as peripheral cores. The system correct behavior is checked by running carefully crafted programs and elaborating specific data. The test programs need to excite the widest part of the system components and, remarkably, to propagate fault effects to observable points, such as primary outputs or memories.

In the automotive field, SBST is a viable solution to the requirements mandated by standards [1] to increase the dependability of microprocessor-based embedded systems. In simple words, the microprocessor is periodically forced to execute the self-test code able to detect the possible permanent faults appearing during the product mission time. In this approach, self-test code is uploaded in a non-volatile memory during the final device test; when run, such a program writes the self-test results in a specific memory space, then they are read out through an available serial port by an external test controller that inhibits the device operations in case of an erroneous response. Fig. 1 shows the principle of the approach.

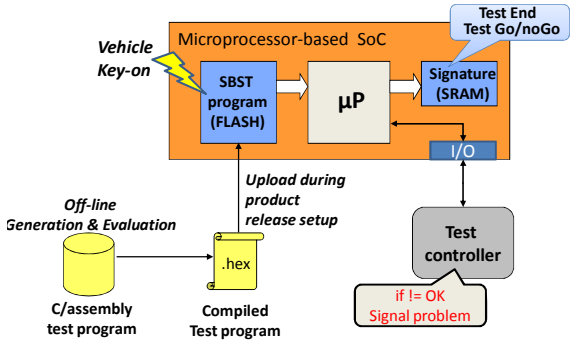

Figure 1. In-Field Self-Test procedure execution.

III. Fault Grading of In-Field SBST Procedures

Fault grading of a SBST procedure corresponds to the task of measuring its ability in detecting the faults possibly affecting the entire SoC or a part of it (*fault coverage*). This process is not trivial; in fact, the generic fault grading process flow, which is illustrated in Fig. 2, may become slow or inaccurate if some important factors are neglected. A fault grading procedure is traditionally composed of three consecutive phases.

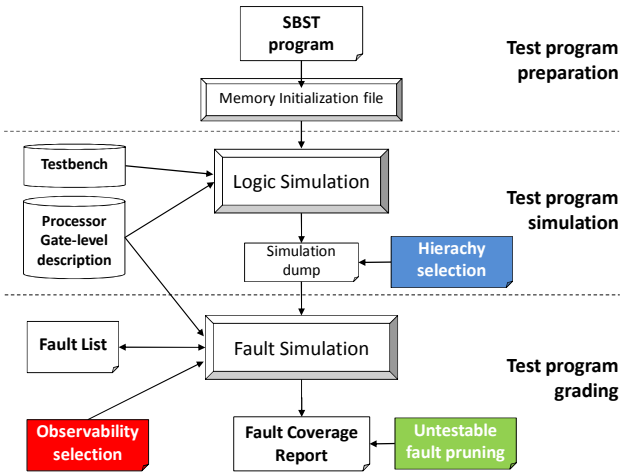

Figure 2. Fault grading flow resulting in a coverage report.

The first phase is the SBST program preparation. We suppose that a test program is already available (e.g., developed by the system designers): here, it has to be compiled and converted in a format compatible with the simulation and fault simulation environments.

Secondly, the circuit has to be simulated having care of mimicking the device mission behavior. The compliancy with the operative field conditions, such as configuration at reset time, has to be achieved using a suitable testbench. In case the execution of the SBST program is preceded by generic initialization of memory cores, this part possibly has to be skipped by using the simulator abilities to acquire memory images; this feature permits significant time reduction during both simulation and fault simulation. The result of the simulation step is the simulation dump file, which is later used for providing inputs to fault simulation. Depending on the SoC part whose fault coverage has to be measured, an appropriate system hierarchy level has to be selected in order to save time in the fault simulation process. At the selected hierarchy level it is possible to effectively observe faulty behavior as well as at the device top level.

The final step of the SBST procedure evaluation consists in the fault simulation process; during this phase, a fault list is generated and processed, based on the SoC gate level description at the selected system hierarchy level. Depending on the available EDA tools, manipulation of the simulation dump may be needed to assure compatibility with the fault simulator. To achieve fair fault coverage figures, the signals to be observed to distinguish among correct and corrupted behavior have to be properly selected. This is a crucial point in the entire flow, since an inaccurate selection may lead to imbalanced measurements. If an inner hierarchy level has been selected during the simulation phases, all signals involved in the transport of fault effects outside the selected circuit zone and observable by upper levels have to be counted for observation, while the others have to be left out. Furthermore, these signals have to be observed only during significant operations such as when the test signature is transmitted.

At the end of the grading process a fault coverage report is produced. To show effective fault coverage, this report should be further manipulated to remove the set of faults which cannot be detected by means of a SBST procedure. As an example, signals belonging to scan chains used for manufacturing test may become faulty but never affect the in-field SoC behavior, or modules used to perform software debug will never be used during the mission, thus can be skipped in the grading process.

IV. Case Study

A SoC based on a 32-bit pipelined processor was used as a case study for the shown fault grading flow, and a set of 9 SBST program were evaluated. The focus of the fault grading process was put on the SoC part called *platform* which includes the principal computing elements of the SoC. Fig. 3 shows the SoC architecture and the selected hierarchy level (in blue), the observed outputs (in red) and some of the parts excluded from the fault simulation because not used in functional circuit operations (in green). Table I summarizes the figures concerning the fault grading flow; the purpose of such flow was obtaining the Stuck-At (SA) fault coverage measure of the 9 SBST programs, whose length is 4,142 clock cycles. Test program code and content of the several memory cores at key-on were purposely forced to known values by the fault simulator prior to performing the simulation step.

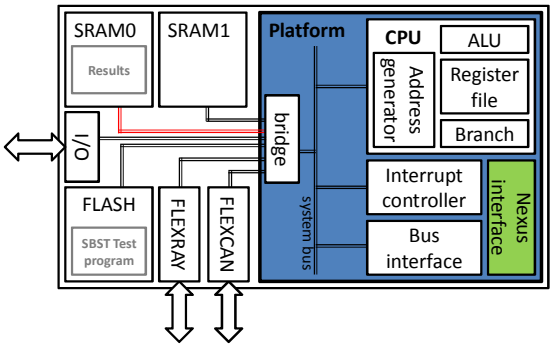

Figure 3. SoC architecture with hierarchy selection (in blue), observability selection (in red) and fault pruning zones (in green).

TABLE I. Figures of the Experimented Fault Grading Process

SA faults [#]	*Program preparation time*	*Logic simulation CPU time*	*Fault simulation CPU time*
~504K	<< 1 s	2.15 m	27.3 h

References

[1] ISO/DIS 26262 "Road vehicles – functional safety", 2009 (under development).

[2] A. Paschalis and D. Gizopoulos, "Effective software-based self-test strategies for on-line periodic testing of embedded processors", IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 24, n. 1, Jan. 2005, pp. 88 – 99.